

Modo Real (16 bits)

- programación en 16 bits, solo se puede programar para este modo en ASM.
- todas las instrucciones están disponibles; AX, CX y DX no son de propósito general.
- registros de segmento (CS (código), SS (segmento), DS (data default), ES GS FS(datos))

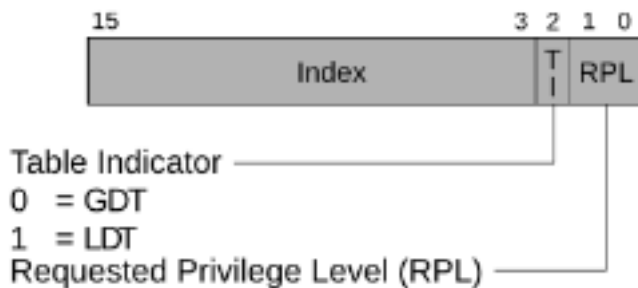
DIRECCIONAMIENTO:

Lógica --- (segmentación) ---> **Lineal** --- (paginación) ---> **Física**

Dirección lógica = SEL:OFF

donde SEL es selector de segmento (índice de una entrada en una tabla de descriptores de segmento) del cuál se obtiene el base address al que se le sumará el OFFset para formar la dirección LINEAL.

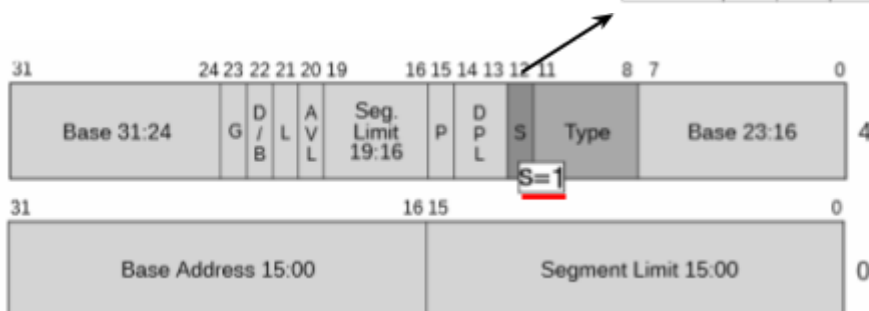
Selector de Segmento:



Decimal	Type Field				Descriptor Type	Description
	11	10 E	9 W	8 A		
0	0	0	0	0	Data	Read-Only
1	0	0	0	1	Data	Read-Only, accessed
2	0	0	1	0	Data	Read/Write
3	0	0	1	1	Data	Read/Write, accessed
4	0	1	0	0	Data	Read-Only, expand-down
5	0	1	0	1	Data	Read-Only, expand-down, accessed
6	0	1	1	0	Data	Read/Write, expand-down
7	0	1	1	1	Data	Read/Write, expand-down, accessed
		C	R	A		
8	1	0	0	0	Code	Execute-Only
9	1	0	0	1	Code	Execute-Only, accessed
10	1	0	1	0	Code	Execute/Read
11	1	0	1	1	Code	Execute/Read, accessed
12	1	1	0	0	Code	Execute-Only, conforming
13	1	1	0	1	Code	Execute-Only, conforming, accessed
14	1	1	1	0	Code	Execute/Read, conforming
15	1	1	1	1	Code	Execute/Read, conforming, accessed

obs: en el tp3 usamos tipo 2 para datos y 8 para código

Descriptor de segmento:



conforming:
"ajustan" su
nivel de
privilegio

- L — 64-bit code segment (IA-32e mode only)
- AVL — Available for use by system software
- BASE — Segment base address
- D/B — Default operation size (0 = 16-bit segment; 1 = 32-bit segment) → 1
- DPL — Descriptor privilege level
- G — Granularity → limite indica cantidad de 0 = byte, 1 = 4kb que direcciona el segmento
- LIMIT — Segment Limit
- P — Segment present
- S — Descriptor type (0 = system; 1 = code or data)
- TYPE — Segment type

(Si s=1 es la tabla de arriba, si s=0 es la tabla "system-segment and Gate-descriptor types (fig 3-2 vol 3. 3-19)
(con G=1 se puede direccionar hasta 4gb)

(LOS REGISTROS CR SE PUEDEN VER CON `mov eax, crN` y escribir con `mov crN, eax` donde n es un num de CR (x ej, cr3))

Modo protegido: (32 Bits)

→ Antes de pasar a modo protegido:

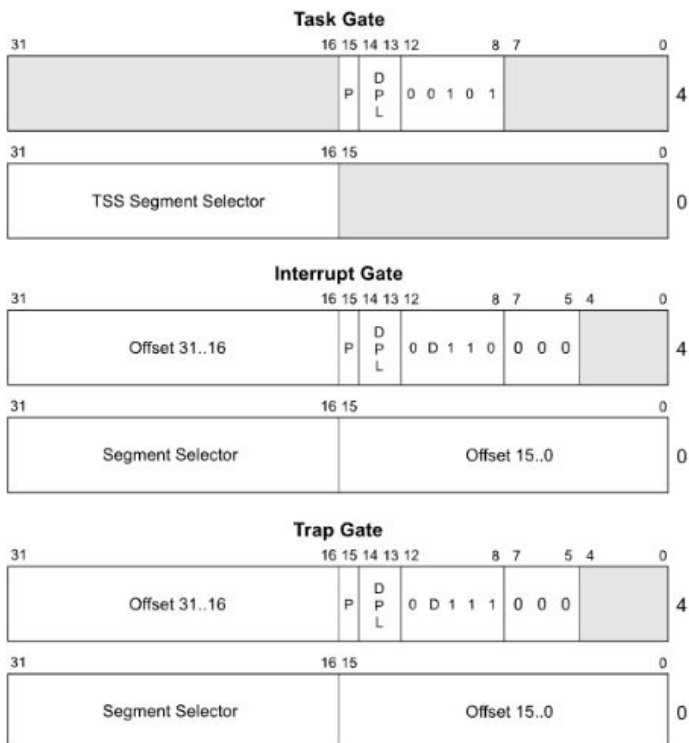
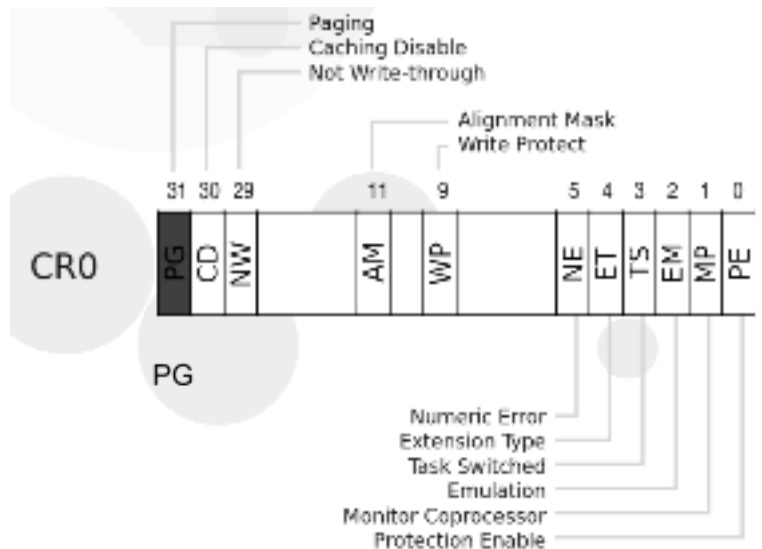
- ◆ Deshabilitar interrupciones (CLI)
- ◆ habilitar A20..?
- ◆ Hay que cargar el registro GDTR con la instru `LGDT` (guarda la dirección de la GDT). La GDT debe tener **al menos** un descriptor nulo, uno de código y uno de datos.
- ◆ Prender bit PE de C0
- ◆ FAR JMP a la sgte instru (`JMP CS:etiqueta sgte instru`)

Sgte instrucción: `CS:EIP` (si escribo código de kernel, x ej, uso `cs = seg código de kernel`)

- ◆ Cargar los reg de segmento (excepto CS) (establecer SS + setear ESP)

Interrupciones:

En la IDT se guardan los Interrupt Descriptors. En IDTR se guardan la dirección base de la IDT (bits 47:16) y el límite de la IDT (15:0, última pos direccionable de la tabla. $n \cdot 8 - 1$). Descriptores de IDT:



DPL Descriptor Privilege Level
Offset Offset to procedure entry point
P Segment Present flag
Selector Segment Selector for destination code segment
D Size of gate: 1 = 32 bits; 0 = 16 bits

Reserved

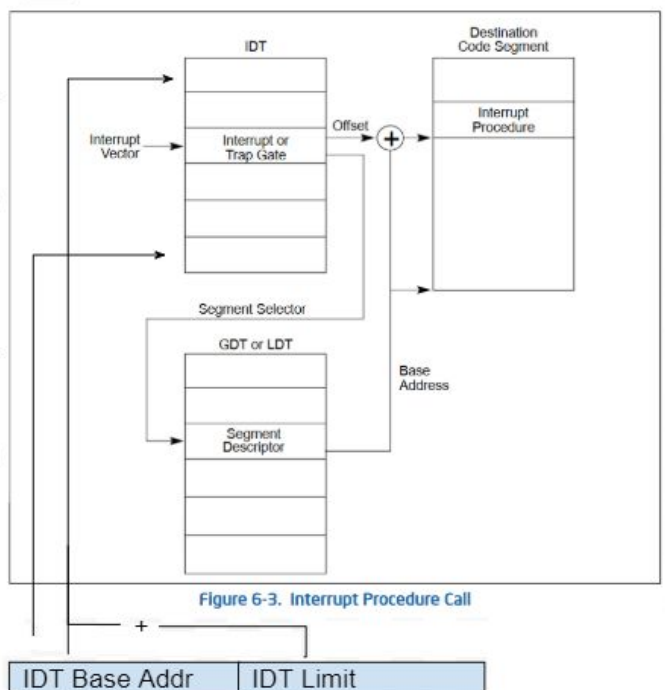
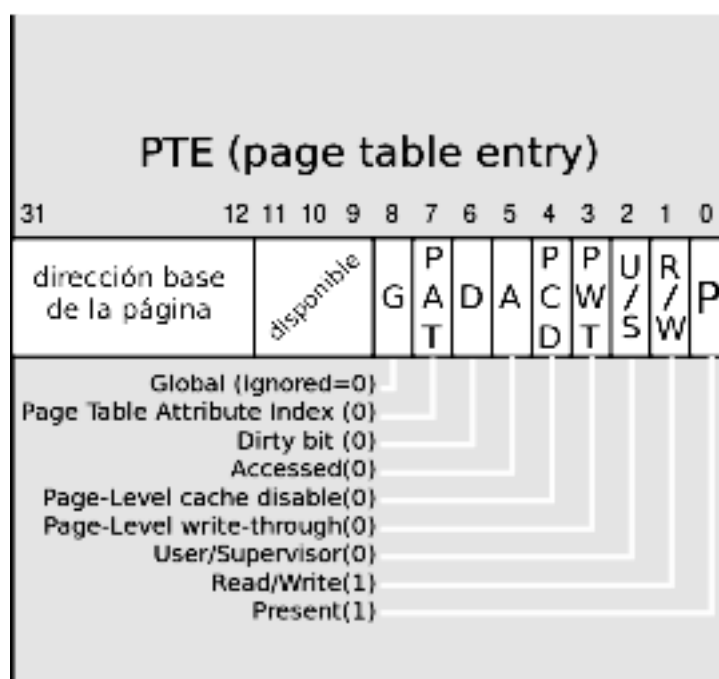
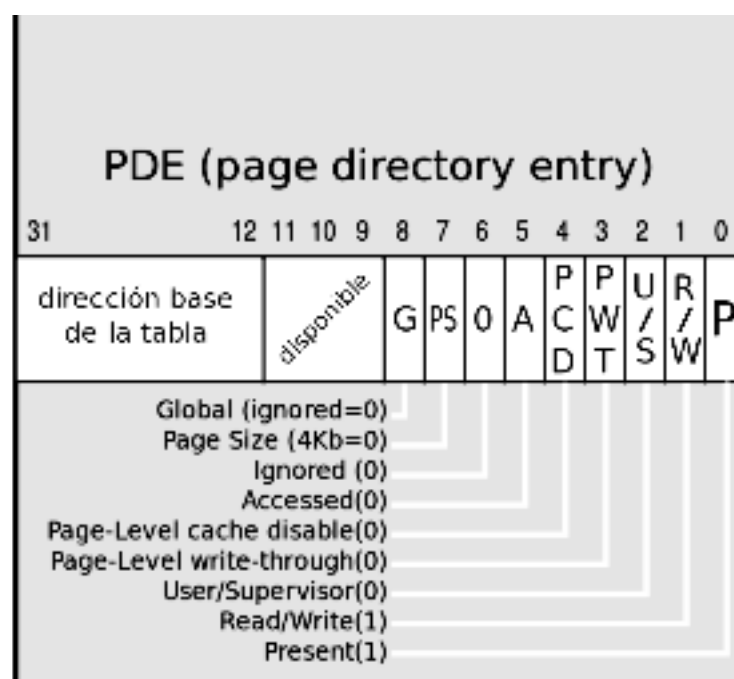
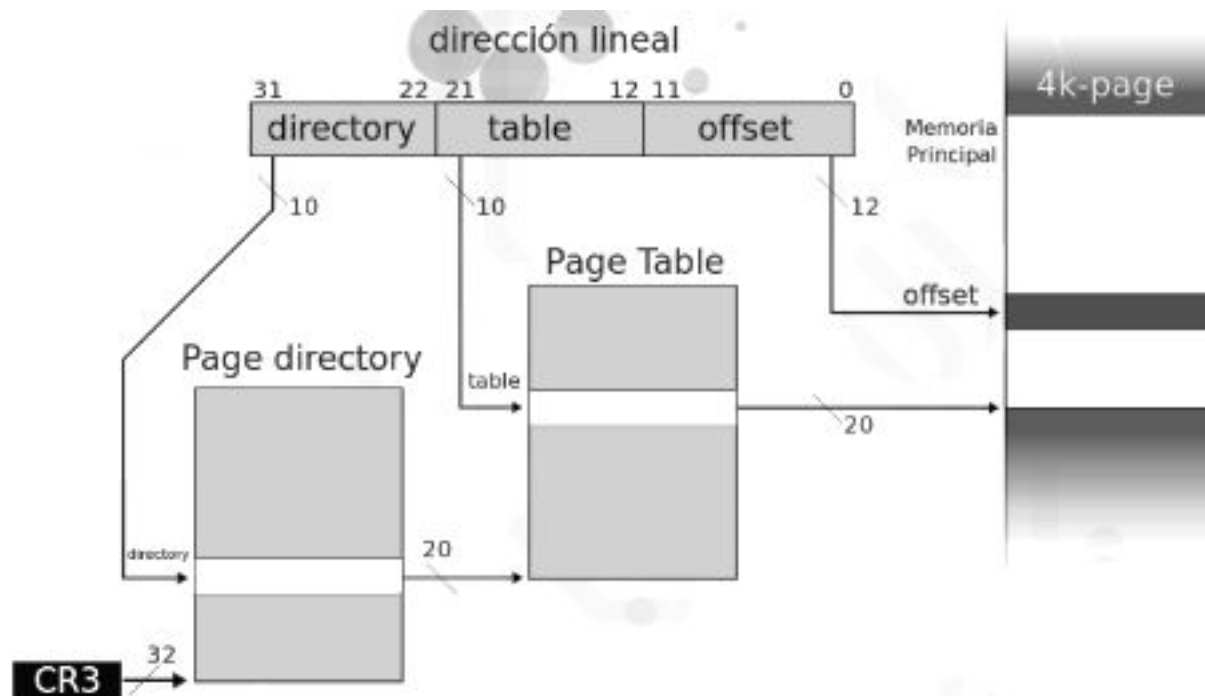


Figure 6-3. Interrupt Procedure Call

(recordar: los parametros se pasan de atras p adelante en el stack)

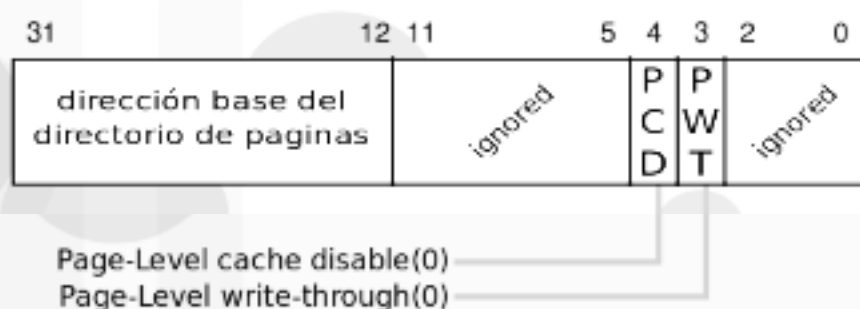
(rutina de permutación de tarea de reloj en la hoja de interrupciones, no entraba acá)

Paginación:



CR3

- Armar un directorio de páginas y tablas de páginas
- Poner en CR3 la dirección base del directorio de páginas
- Limpiar bits PCD y PWT de CR3
- Setear el bit PG de CR0



identity mapping: la dir lineal y la física son la misma.

KB = 2^{10} ; MB = 2^{20} ; GB = 2^{30} BYTES

supervisor: el acceso es explícito (es decir lo hacés mediante una instrucción), entonces supervisor es 0, 1 y 2 y usuario es 3 en paginación. Es decir, el nivel de usuario se corresponde **solo** con nivel **3** de segmentación

Procedimiento para mapear una página

1. Descomponemos la **dirección virtual** en **índice del page directory** y en **índice del page table**.
2. Utilizar la dirección del **page directory** y el **índice del page directory** para encontrar el **page directory entry** asociado al índice.
 - Hay que chequear que la page table a la que referencia el **pde** exista.
 - Si no existe, hay que crearla y setear correctamente (bits de propiedades) la **pde** para que pueda ser accedida posteriormente.
3. Utilizar el **índice del page table** para obtener la **page table entry** correspondiente.
4. Completar la **pte** para que reference a la **dirección física**.
5. Ejecutar **tlbflush()** para invalidar la cache de traducciones.

Procedimiento para desmapear una página

1. Descomponemos la **dirección virtual** en **índice del page directory** y en **índice del page table**.
2. Utilizar la dirección del **page directory** y el **índice del page directory** para encontrar el **page directory entry** asociado al índice.
3. Utilizar el **índice del page table** para obtener la **page table entry** correspondiente.
4. Establecer el bit de presente en 0.
5. **tlbflush()**

obs: cambiar el CR3 (requiere privilegio 0) invalida automáticamente las entradas de la tlb no globales

Stack usage:

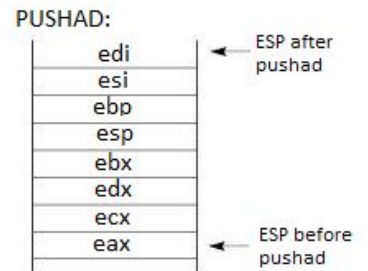
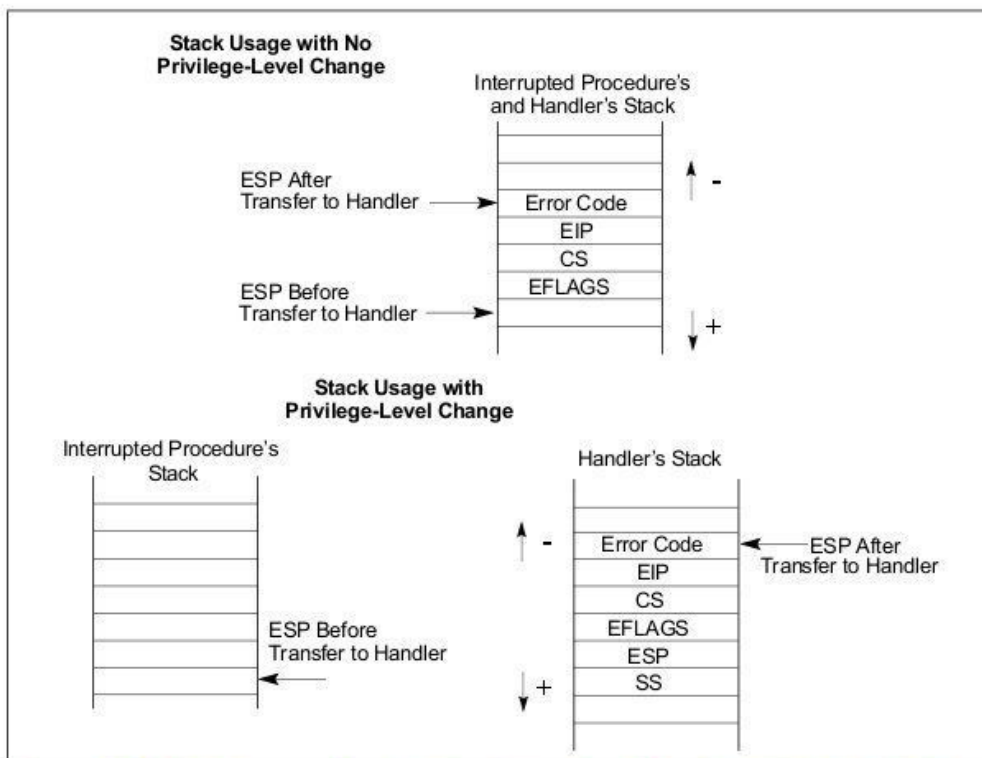


Figure 5-4. Stack Usage on Transfers to Interrupt and Exception-Handling Routines

Macro exception handler:

`_isr%1:`

```

pushad
push ds
push es
push fs
push gs
push dword %1
call idt_exception_handler
add esp, 16
pop ad

```

Tareas:

Unidades de trabajo q el procesador puede despachar, ejecutar y suspender. Se suelen usar para ejecutar una instancia de un programa. La arquitectura provee mecanismos para salvar el estado de una tarea, despacharla para su ejecución y conmutar tareas.

Componentes:

1. Espacio de ejecución:
 - a. Segmento de código
 - b. Seg de datos/pila (uno o varios)
2. Segmento de estado (**Task State Segment**):
 - a. almacena el contexto de la tarea para poder reanudarla en el mismo lugar

El TR contiene selector de segmento de TSS de la tarea q se ejecuta ahora.

(Ver TSS y descriptor de TSS en resumen manual intel)

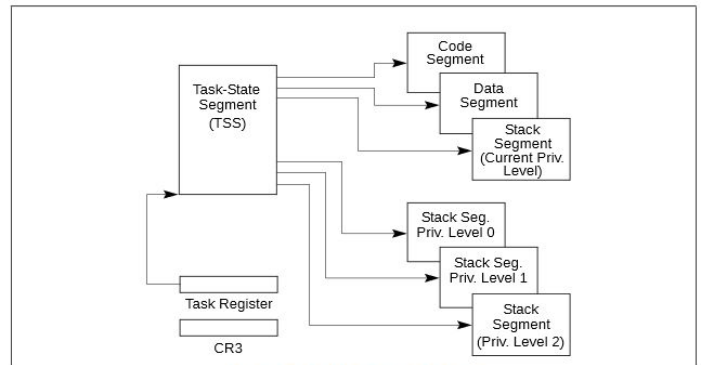
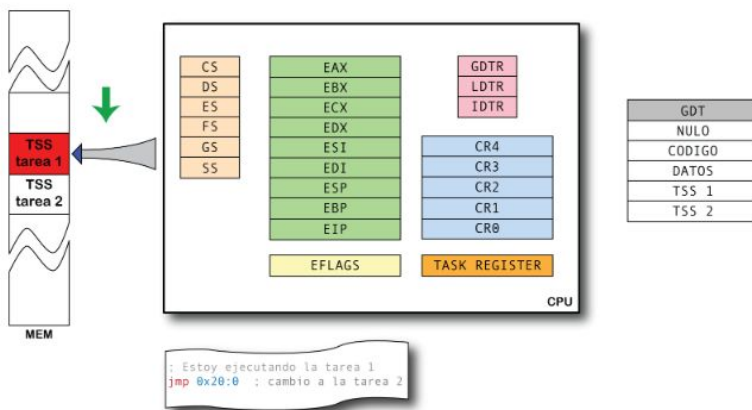


Figure 7-1. Structure of a Task



Conmutacion de tareas:

1. `jmp 0x20:0 (TSS 2)`
2. busco descriptor correspondiente en GDT
3. como cambio tarea, leo TR
4. busco TSS apuntado por TR
5. guardo contexto (imagen)
6. Busco TSS apuntado x descriptor de tarea a ejecutar
7. obtengo nuevo contexto (CR3+LDTR+registros generales y de segmento y flags)
8. Actualizo TR

9. Continúa ejecución con nuevo contexto

Siempre que se salta a una tarea hay cambio de contexto. Antes de saltar se guarda el contexto actual. Por eso hace falta una tarea inicial, provee una TSS donde el procesador puede guardar el contexto actual. Ese es su único propósito.

1. Al iniciar las tareas:

- ▶ completar **EIP**
- ▶ completar **ESP** y **EBP**
- ▶ completar selectores de segmento
- ▶ completar **CR3**
- ▶ completar **EFLAGS** (*)

*EFLAGS por defecto : 0x00000002

EFLAGS con interr habilitadas: 0x00000202

2. Al saltar por primera vez a una tarea:

- ▶ tener un descriptor en la GDT de la tarea inicial
- ▶ tener un descriptor en la GDT de la tarea a saltar
- ▶ tener en **TR** algun valor válido (tarea inicial)

Protección: En segmentación

(Ver graficos descriptor de selector de segmento (x RPL) y de GDT)

■ ¿Cómo sabemos si el segmento seleccionado está presente?

Vemos si el bit P de su respectivo descriptor está en 1

■ ¿Y para saber si la operación no supera el límite del segmento?

- Si el bit de granularidad (G) es 0:

$\text{offset} + \text{bytes a referenciar} - 1 \leq \text{límite}$

- Si no, tenemos varias alternativas:

- $\text{offset} + \text{bytes a referenciar} - 1 \leq ((\text{límite} + 1) * 4\text{kb}) - 1$
- $\text{offset} + \text{bytes a referenciar} - 1 \leq ((\text{límite} + 1) << 12) - 1$
- $\text{offset} + \text{bytes a referenciar} - 1 \leq (\text{límite} << 12) + 0\text{xFFF}$

■ ¿Qué sucede si alguna de las verificaciones anteriores no se cumple? → General Protection Fault (#GP)

- Si el segmento es de **datos** → $\text{EPL} \leq \text{DPL}$

- Si es un segmento de **código** tenemos dos casos:

- Non-conforming → $\text{EPL} = \text{DPL}$
- Conforming → $\text{CPL} \geq \text{DPL}$ (Ojo que no se usa el EPL)

Nota: Las comparaciones $=$, \geq , \leq son por valor numérico

■ ¿Qué sucede si alguna de las verificaciones anteriores no se cumple? → General Protection Fault (#GP)

■ ¿Qué tipo de segmentos nos permiten leer, escribir o ejecutar?

Los que tienen el bit de sistema (S) en 1

■ ¿Cuáles nos permiten realizar lecturas?

Los de datos y los de código con lectura habilitada

■ ¿Y escrituras?

Los de datos con escritura habilitada

■ ¿Y en el caso de las ejecuciones?

Los de código

■ ¿Qué sucede si alguna de las verificaciones anteriores no se cumple? → General Protection Fault (#GP)

DPL: (Descriptor Privilege Level)
nivel de privilegio del segmento a ser accedido (está en su descriptor)

CPL: (Current Privilege Level) DPL del segmento de código en ejecución

RPL: (Requested Privilege Level)
nivel de privilegio marcado x los dos bits menos significativos del selector de segmento. Se puede cambiar!!

EPL: (Effective Privilege Level)
 $\text{max}(\text{CPL}, \text{RPL})$

En paginación: (ver graficos PTE y PDE)

■ ¿Cómo sabemos si la página a acceder está presente?

Vemos el bit de presente (P) de su PTE

■ ¿Y para saber si la tabla de la PTE lo está?

Vemos el bit P de la PDE que la referencia ¿Puede suceder que el directorio no esté presente al tener paginación habilitada?

■ ¿Qué simboliza el bit R/W de las entradas?

El bit Read/Write indica:

- 0: Sólo hay permisos lectura (Read Only)
- 1: Hay permisos de lectura y escritura (Read Write)

■ ¿Y el bit U/S?

El bit User/Supervisor indica:

- 0: Sólo hay permisos de acceso para nivel supervisor (Supervisor)
- 1: Hay permisos de acceso para los niveles supervisor y usuario (User)

Nota: El nivel usuario se corresponde sólo con el nivel 3 de segmentación

Ahora, suponiendo que las entradas y la página están presentes...

-> si no, page fault

■ ¿Qué páginas permiten escrituras?

Aquellas cuyas PDE y PTE tengan R/W en 1

■ ¿Y lecturas?

Todas las combinaciones lo permiten

■ ¿Qué páginas pueden ser accedidas por nivel usuario?

Aquellas cuyas PDE y PTE tengan U/S en 1

■ ¿Y supervisor?

Todas las combinaciones lo permiten ¿Podría el nivel supervisor escribir en una página de sólo lectura?

■ ¿Qué sucede si alguna de las verificaciones anteriores no se cumple? → Page Fault (#PF)

-> Depende de bit WP (write protect): "when set, inhibits supervisor-level procedures to write into read-only pages (regardless of the U/S bit setting)"

Interrupciones: (ver graficos gates)

■ ¿Cómo sabemos si la entrada de una interrupción está presente?

Vemos el bit P de su descriptor en la IDT

■ ¿Y para saber en qué segmento se ejecuta su RAI?

Usamos el selector de segmento de su descriptor en la IDT

■ ¿Cómo sabemos si tenemos el nivel de privilegio necesario para llamar a una interrupción (instrucción INT)?

Verificamos que $CPL \leq DPL$ siendo estos los bits de su descriptor en la IDT

■ ¿Qué sucede con las interrupciones de hardware?

Los bits de DPL se ignoran

■ ¿Qué sucede si alguna de las verificaciones anteriores no se cumple? → General Protection Fault (#GP)

Tareas: (ver grafico descriptor TSS)

Nos enfocamos en ver si podemos conmutar a una tarea dada viendo su descriptor de TSS

■ **¿Cómo sabemos si la información de una tarea está presente?**

Vemos que si el bit P del descriptor de su TSS está en 1

■ **¿Y qué nivel de privilegio es necesario para saltar a ella?**

Para realizar el salto se necesita que $CPL \leq DPL$ donde estos son los bits del descriptor de su TSS. ¿Eso significa que no se puede conmutar a una tarea de nivel 0 al ejecutarse una de nivel 3?

■ **¿Se puede saltar a una tarea si B es 1?**

No, es necesario que el bit de Busy esté en 0

■ **¿Qué sucede si alguna de las verificaciones anteriores no se cumple?** → General Protection Fault (#GP)

Algunas instrucciones útiles:

cli = deshabilitar interrupciones

sti = habilitar interrupciones

LGDT = cargar la gdt (toma como parámetro dir de GDT)

LIDT, LLDT = mismo q lgdt, con IDT y LDT respectivamente

mov eax, cr# y mov cr#, eax para descargar y cargar registros CR# (como cr3)

ltr ax = cargar ax en tr

str ax = cargar tr en ax (para poder leerlo yo)

pic_finish1

tlbflush

pic_reset (antes de usar pic!)

pic_enable/disable

Protected Mode Exceptions

#GP(0)	<p>If offset in target operand, call gate, or TSS is beyond the code segment limits.</p> <p>If the segment selector in the destination operand, call gate, task gate, or TSS is NULL.</p> <p>If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.</p> <p>If the DS, ES, FS, or GS register is used to access memory and it contains a NULL segment selector.</p>
#GP(selector)	<p>If the segment selector index is outside descriptor table limits.</p> <p>If the segment descriptor pointed to by the segment selector in the destination operand is not for a conforming-code segment, nonconforming-code segment, call gate, task gate, or task state segment.</p> <p>If the DPL for a nonconforming-code segment is not equal to the CPL.</p> <p>(When not using a call gate.) If the RPL for the segment's segment selector is greater than the CPL.</p> <p>If the DPL for a conforming-code segment is greater than the CPL.</p> <p>If the DPL from a call-gate, task-gate, or TSS segment descriptor is less than the CPL or than the RPL of the call-gate, task-gate, or TSS's segment selector.</p> <p>If the segment descriptor for selector in a call gate does not indicate it is a code segment.</p> <p>If the segment descriptor for the segment selector in a task gate does not indicate an available TSS.</p> <p>If the segment selector for a TSS has its local/global bit set for local.</p>
#SS(0)	<p>If a TSS segment descriptor specifies that the TSS is busy or not available.</p> <p>If a memory operand effective address is outside the SS segment limit.</p>
#NP (selector)	<p>If the code segment being accessed is not present.</p> <p>If call gate, task gate, or TSS not present.</p>
#PF(fault-code)	<p>If a page fault occurs.</p>
#AC(0)	<p>If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3. (Only occurs when fetching target from memory.)</p>
#UD	<p>If the LOCK prefix is used.</p>

Tipo	Mnemónico	Descripción	Clase	Código de Error	Fuente
0	#DE	Error de División	Fault	NO	Instrucciones DIV e IDIV
1	#DB	Reservada	Fault/Trap	NO	Solo para uso de Intel
2	-	NMI	Interrupción	NO	Interrupción No enmascarable. Pin NMI
3	#BP	BreachPoint	Trap	NO	Opcodex0xCC
4	#OF	Overflow	Trap	NO	Instrucción INTO
5	#BR	BOUND Range Exceeded	Fault	NO	Instrucción BOUND
6	#UD	Invalid Opcode	Fault	NO	Instrucción UD2 u Opcode Reservado
7	#NM	Coprocesador No disponible	Fault	NO	Instrucciones de Punto Flotante o WAIT / FWAIT
8	#DF	Doble Fault	Abort	SI (Cero)	Cualquier instrucción capaz de generar una excepción, una señal en NMI o en INTR
9		Coprocessor Segment Overrun (reservada)	Fault	SI	Instrucciones de Punto Flotante
10	#TS	TSS Inválido	Fault	SI	Task switch o acceso a un TSS
11	#NP	Segmento No Presente	Fault	SI	Carga o acceso a un registro de segmento

Tipo	Mnemónico	Descripción	Clase	Código de Error	Fuente
12	#SS	Falta en el Stack Segment	Fault	SI	Operaciones de Pila y Carga del registro SS
13	#GP	General Protection	Fault	SI	Cualquier referencia a memoria y otros chequeos de protección
14	#PF	Page Fault	Fault	SI	Cualquier referencia a memoria
15	-	Reservada por Intel (no usar)		NO	
16	#MF	X-87 FPU Error de Punto Flotante	Fault	NO	Instrucción de la FPU o WAIT/FWAIT
17	#AC	Alignment Check	Fault	SI (Cero)	Cualquier referencia de datos a memoria
18	#MC	Machine Check	abort	NO	Los Códigos de error si hubiese, así como su fuente, depende del modelo de procesador
19	#XF	Excepción SIMD Floating Point	Fault	NO	Cualquier instrucción SSEx
20-31	-	Reservada por Intel (no usar)			
32-255	-	A definir por el usuario	Interrupción		Interrupciones externas o Instrucción INTn

(en general 32 = reloj, 33 = teclado. NO USARLAS PARA OTRA COSA)

Permutación de tarea en reloj:

offset: dd 0

selector: dw 0

_isr32:

```

pushad
call pic_finish1
call sched_nextTask
str cx
cmp ax, cx
je .fin
    mov [selector], ax    -> en ax esta el selector de la GDT de TSS a la q saltar.
    jmp far [offset]      Se elige con alguna función que dé la tarea siguiente.
                          (en este caso, sched_nextTask)
.fin:
popad
iret

```